第3章　指　　针

3.1　内容介绍

指针是用来控制对象的内存地址，它的功能非常强大，可以直接访问和操作系统内存，合理地运用指针也会让程序的性能得到很好的优化。

C语言中指针的引入是为了便于直接与硬件进行交互，通常把能够表示内存单元的地址称为指针。指针是C语言中的一个重要概念，正确而灵活地运用指针，可以有效地表示复杂的数据结构；能动态分配内存；方便地使用字符串；能直接处理内存单元地址等。

本章主要介绍内存地址、变量地址、变量指针的概念，以及指针变量的使用，指针的运算、动态内存分配等内容。

注意：在本章实例中将输出一些地址，不同系统或不同的运行环境，地址的值可能会有一定的不同，因此本章输出的地址只作为参考。

3.2　实　　例

实例3.2.1　输出变量的地址

1．问题描述与分析

编写程序，在程序中定义一些变量，然后输出这些变量的地址。

在程序运行时，都会为程序中定义的变量分配内存单元，而内存单元是按一定的规律编号的，这个编号也称为内存单元的地址。内存通常是以字节为单位进行编号的，不同类型的变量所占用的内存单元数是不一样的，例如，字符型变量占用1个内存单元，整型变量占用4个内存单元，人们称变量所占用的起始单元的地址为变量的地址。可以使用符号&获取某个变量的地址。

2．程序及运行结果

```
#include <stdio.h>
main()
{
    int a=10;
    float b=20.6f;
    char c='A';
    printf("%d, %5.2f, %c\n", a, b, c);
    printf("%d, %d, %d\n", &a, &b, &c);
    printf("%x, %x, %x\n", &a, &b, &c);
}
```

运行结果如下：

```
10, 20.60, A
1245052, 1245048, 1245044
12ff7c, 12ff78, 12ff74
```

其中第一行是输出变量的值,第二行以十进制输出变量的地址,第三行以十六进制输出变量的地址。

3. 知识点

1) 内存地址

计算机的内存储器被划分成一个个的存储单元,这些存储单元按一定的规则编号,这个编号就是存储单元的地址。地址编码的最基本单位是字节(用 B 表示),每个字节由 8 个二进制位组成,因此一个最基本内存单元的大小就是一个字节。

每个存储单元都有一个唯一的地址,在这个单元中可以存放指定的数据。

2) 变量的地址

在程序中定义的所有变量,在内存中都要分配相应的存储单元,不同类型的数据所需要的存储空间的大小不同。例如,在 Visual C++ 6.0 中,字符型数据占 1B,短整型数据占 2B,整型数据占 4B。

系统分配给变量的内存空间的起始单元地址称为该变量的地址,为了方便,一般用十六进制数表示内存单元或变量的地址。如程序中定义的字符型变量 c,就分配 1B 的内存单元,其地址是 0x12ff74(以 0x 开头的整数是十六进制数),而 a 是一个整型变量,需要分配 4B 的内存空间(这 4B 是连续的 4 个存储单元),这 4 个单元的编号分别为 0x12ff7c、0x12ff7d、0x12ff7e 和 0x12ff7f,称这 4 个连续存储单元的起始单元的编号 0x12ff7c 为整型变量 a 的地址。同样,程序中的变量 b 是单精度实型变量,也需要分配 4B 的连续内存空间,这 4 个单元的编号分别为 0x12ff78、0x12ff79、0x12ff7a 和 0x12ff7b,称这 4 个连续存储单元的起始单元的编号 0x12ff78 为实型变量 b 的地址。变量 a、b、c 占用内存的情况可以用图 3.1 来表示(左侧的一列是存储单元的地址)。

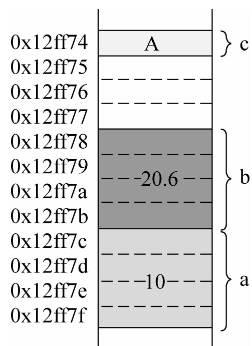

图 3.1 变量在内存中占用的存储单元及编号

图 3.1 中第一个颜色较浅的 1 个单元是变量 c 所占用的内存单元,然后的 3 个内存单元没有被使用,再后面的 4 个颜色较深的单元是变量 b 所占用的内存,最后 4 个颜色较浅的单元是变量 a 所占用的内存。为什么变量 b 和变量 c 所占内存之间有 3 个单元没有利用,而造成一定的内存浪费,这是因为 Visual C++ 6.0 编译器为了优化数据的存取效率,采用了内存对齐的策略。

3) 变量的指针

一个变量的地址也称为该变量的指针。例如,变量 a 的地址为 0x12ff7c,称地址 0x12ff7c 为变量 a 的指针,因此在 C 语言中,变量的指针与变量的地址是同一个概念。

可以使用符号 & 获取变量的地址。

实例 3.2.2　通过指针变量存取变量的值

1. 问题描述与分析

不仅可以定义保存整数、实数等基本数据类型的变量，C也可以定义专门用于保存指针（地址）的变量，称保存指针的变量为指针变量。可以在程序中通过指针变量间接地访问变量的值。

在本实例程序中定义基本数据类型变量a、b、c，以及对应的指针变量p1、p2、p3，使用指针变量访问变量a、b、c的值，并输出变量的值以及变量的地址。

2. 程序及运行结果

```
#include <stdio.h>
main()
{
    int a, *p1;
    double b, *p2;
    char c, *p3;
    p1=&a;
    p2=&b;
    p3=&c;
    *p1=10;
    *p2=11.2;
    *p3='A';
    printf("%4d %4d %x %x %x\n", a, *p1, &a, p1, &p1);
    printf("%4.1f %4.1f %x %x %x\n", b, *p2, &b, p2, &p2);
    printf("%4c %4c %x %x %x\n", c, *p3, &c, p3, &p3);
}
```

运行结果如下：

```
10    10    12ff7c  12ff7c  12ff78
11.2  11.2  12ff70  12ff70  12ff6c
A     A     12ff68  12ff68  12ff64
```

3. 知识点

1）指针变量的定义

以前我们使用过整型变量，用于存储整数，也使用过实型变量，用于存储实型数据。同样也可以定义一个专门用于存储其他变量的指针（即地址）的变量，称这种变量为指针变量（当然也可以称为地址变量）。

指针变量的定义如下：

```
数据类型    *指针变量名;
```

例如：

```
int  a, *p1;
double  b, *p2;
```

```
char  c, *p3;
```

以上第一行定义一个整型变量 a 和一个整型指针变量 p1,指针变量 p1 只能保存整型变量的地址(指针);第二行定义一个实型变量 b 和一个实型指针变量 p2,指针变量 p2 只能保存实型变量的地址(指针);第三行定义一个字符型变量 c 和一个字符型指针变量 p3,指针变量 p3 只能保存字符型变量的地址(指针)。

由上面的例子可以看出,与一般变量的定义相比,除变量名前多了一个星号(*)外,其余都一样。

2) 指针运算符(*)与取地址运算符(&)

对于指针变量有两个有关的运算符:指针运算符(*)和取地址运算符(&)。

本实例程序定义了 3 个指针变量,其中 p1 定义为保存整型变量的地址,然后将 a 的地址(使用了取地址运算符 &)赋给指针变量 p1,我们称指针 p1 指向变量 a。同样 p2 指向变量 b,p3 指向变量 c。图 3.2 显示了 3 个指针变量与 3 个变量的关系。

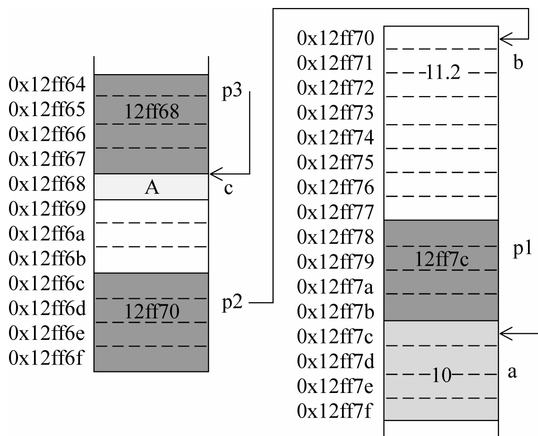

图 3.2 三个指针分别指向三个变量

指针变量 p1 保存的是变量 a 的地址 0x12ff7c,*p1 表示地址 0x12ff7c 处的整数值,也就是 a 的值,*p1=10 表示将 p1 所保存的地址那个内存单元赋值为 10(即将 10 赋给 p1 所指向的内存单元),也就是将 10 赋给变量 a。当然指针变量 p1 本身也占用内存单元,p1 的地址是 0x12ff78。

指针变量 p2 保存的是变量 b 的地址 0x12ff70,*p2 表示地址 0x12ff70 处的双精度实数值,也就是 b 的值,*p2=11.2 表示将 p2 所保存的地址那个内存单元赋值为 11.2(即将 11.2 赋给 p2 所指向的内存单元),也就是将 11.2 赋给变量 b。指针变量 p2 的地址是 0x12ff6c。

指针变量 p3 保存的是变量 c 的地址 0x12ff68,*p3 表示地址 0x12ff68 处的字符,也就是 c 的值,*p3='A'表示将 p3 所保存的地址那个内存单元赋值为'A'(即将'A'赋给 p3 所指向的内存单元),也就是将'A'赋给变量 c。指针变量 p3 的地址是 0x12ff64。

4. 知识补充

1) 指针变量的类型不能改变

指针变量所指向的变量类型是不能改变的,例如,p1 定义为指向整型变量的指针变量,

它就只能指向整型变量(保存整型变量的地址),同样 p2 只能指向 double 型变量,p3 只能指向字符型变量。

2) 指针变量必须有所指向才能引用

指针变量必须指向具体内存位置,才可以引用,否则在运行时可能会发生严重后果。如以下程序:

```
int *p;
*p=10;
```

从语法上看,并没有错误,p 是一个整型指针变量,即保存整型变量的地址,然后为这个地址单元赋值 10。问题是此时 p 还没有具体指向,系统不知道应该将 10 赋给哪个内存单元,如图 3.3 所示。

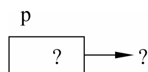

图 3.3 指针没有具体指向之前不能引用

3) 指针变量所占用的字节数

在 Visual C++ 中,指针所占的内存是 4B,可以通过下面的程序验证。

```
#include <stdio.h>
main()
{
    int *p1;
    double *p2;
    char *p3;
    printf("%d  %d  %d  \n", sizeof(p1),sizeof(p2),sizeof(p3));
}
```

运行结果如下:

```
4    4    4
```

实例 3.2.3 指针的算术运算

1. 问题描述与分析

指针可以与整数进行加减运算,也可以进行自增和自减运算。编写程序,定义指针变量,并进行指针变量与整数的加减运算,输出计算结果。

2. 程序与运行结果

```
#include <stdio.h>
void main()
{
    int a, *p1,*p2;
    double b, *p3, *p4;
    p1=&a;
    p3=&b;
    printf("%X    %X\n", p1, p3);
    p2=p1+1;
    p4=p3+1;
```

```
    printf("%X    %X\n", p2, p4);
    p2=p1-1;
    p4=p3-1;
    printf("%X    %X\n", p2, p4);
    p2=p1+5;
    p4=p3+5;
    printf("%X    %X\n", p2, p4);
    p2++;
    p4--;
    printf("%X    %X\n", p2, p4);
    printf("%d    %d\n", p2-p1, p4-p3);
}
```

运行结果如下：

```
0x12FF7C    0x12FF6C
0x12FF80    0x12FF74
0x12FF78    0x12FF64
0x12FF90    0x12FF94
0x12FF94    0x12FF8C
6    4
```

调试程序，分步运行，观察各指针变量值的变化。

3. 知识点

1）指针与整数的加减运算

指针与整数的加减运算实质上是地址的加减运算，对于不同类型的指针，运算结果是不一样的。例如，整型数据占 4B，整型指针加 1 表示当前指针的下一个整数单元的地址，也就是向后 4B，而双精度实型数据占 8B，双精度实型指针加 1 表示当前指针的下一个双精度实数单元的地址，也就是向后 8B。

本实例程序中，变量 a 是整型变量，占 4B，变量 b 是双精度实型变量，占 8B。变量 a、b 占用的内存位置如图 3.4 所示的深色部分。

程序的第 1 行输出的是 p1（a 的地址）和 p3（b 的地址），第 2 行输出的是 p2（p1＋1）和 p4（p3＋1），计算一下 p2 与 p1 的差值为 4（十六进制的 7C 加 4 等于 80），而 p4 与 p3 的差值为 8（十六进制的 6C 加 8 等于 74）。这是因为 p1 是指向整型变量的，整型数据占 4B 的内存空间，而 p3 是指向双精度实型变量的，双精度实型数据占 8B 的内存空间。因此 p1＋1 是下一个整型数据的地址，p2 指向下一个整型数据。p3＋1 是下一个双精度实型数据的地址，即 p4 指向下一个双精度实型数据。

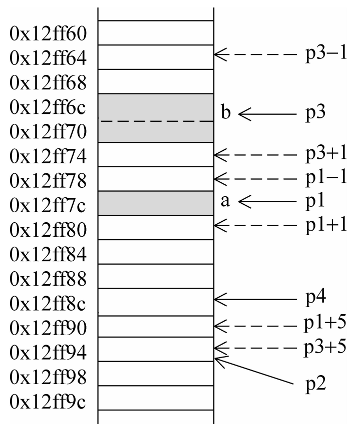

图 3.4　指针与整数进行加减运算的结果

分析第 3 行输出可知，p1－1 是前一个整型数据的地址，即 p2 指向前一个整型数据。p3－1 是前一个双精度实型数据的地址，即 p4 指向前一个双精度实型数据。

同样由第4行输出可知，p1＋5是向后第5个整型数据的地址，即p2指向后面第5个整型数据。p3＋5是向后第5个双精度实型数据的地址，即p4指向后面第5个双精度实型数据。指针与整数加减运算的结果如图3.4所示，注意图中的每个单元代表4B。

由以上分析可以得出，p＋n表示指针p当前所指向位置后面第n个数据的地址，p－n表示指针p当前所指向位置前面第n个数据的地址。

注意：虽然指针可以和整数进行加减运算，但由于加减运算后的地址可能保存程序的其他内容或被操作系统所占用，因此不要对运算完之后的地址进行赋值等操作，否则在程序运行时可能会引起不可预测的问题。在后面的动态分配内存的实例中，以及后面数组一章中，会看到指针变量与整数加减运算的应用。

2）指针变量的自增自减运算

由第5行输出可以看出，指针变量可以进行自增和自减运算，p＋＋与p＝p＋1的作用相同，p－－与p＝p－1的作用相同。

3）指针变量的减法运算

相同类型的指针变量可以进行减法运算，例如，程序的最后一行输出的是两个指针变量相减的结果。图3.4也标明了最终指针变量p1、p2、p3、p4所指向的位置。从运行结果可以看出，两个指针变量的减法表示这两个指针之间相差多少个该种数据的存储空间。例如，(0x12FF94－0x12FF7C)/4＝6，因此p2－p1＝6，(0x12FF8C－0x12FF6C)/8＝4，因此p4－p3＝4。

通常是当两个指针指向同一个数组时，指针变量减法才有意义，有关这部分内容可在下一章的实例中看到。

实例3.2.4　指针的关系运算

1. 问题描述与分析

同一类型的两个指针可以进行关系运算，在程序中定义几个指针变量，并为其赋值，然后输出关系运算的结果。

2. 程序与运行结果

```
#include <stdio.h>
void main()
{
    int a, *p1, *p2, *p3;
    p1=&a;
    p2=p1+10;
    p3=p1;
    printf("%X  %X  %X\n", p1,p2,p3);
    printf("%d  %d  %d\n", p1>p2, p1<p2, p1==p2);
    printf("%d  %d  %d\n", p1>p3, p1<p3, p1==p3);
}
```

运行结果如下：

```
12FF7C  12FFA4  12FF7C
```

```
0  1  0
0  0  1
```

3. 知识点

1）指针的关系运算

指向同一种数据类型的指针可以进行关系运算。如果两个相同类型的指针相等，表示这两个指针指向同一个地址。例如，实例程序中的 p1 和 p3 两个指针变量都指向变量 a，因此两个指针变量相等。而指针变量 p2 等于 p1＋10，因此 p1 小于 p2 为真。3 个指针变量的关系如图 3.5 所示(图中每个单元代表 4B 的内存)。

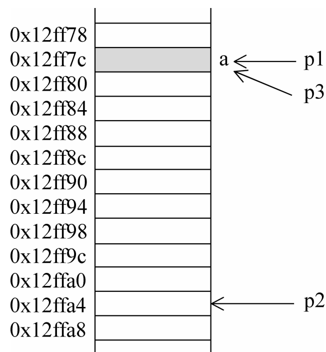

图 3.5　3 个指针变量的关系

2）空指针

为了避免使用空指针，在定义指针变量时，可以将其初始化为 0(也可以写成 NULL)，称之为空指针。在使用指针时可以先判断其是否为空，不为空时才可以使用。如下面的例子。

```
#include <stdio.h>
void main()
{
    int a, *p=NULL;
    if(p!=NULL)
    {
        *p=10;
        printf("将 10 赋值给 p 所指向的地址\n");
    }
    else
        printf("p 是空指针,不能使用! \n");
    p=&a;
    if(p!=NULL)
    {
        *p=10;
        printf("将 10 赋值给 p 所指向的地址\n");
    }
    else
        printf("p 是空指针,不能使用!\n");
}
```

程序运行结果：

```
p 是空指针,不能使用!
将 10 赋值给 p 所指向的地址
```

在遇到第 1 个 if 语句时，p 是 NULL，所以执行 else 部分，输出“p 是空指针，不能使用!”。然后将变量 a 的地址赋给 p，p 已经不是空指针了，所以第 2 次遇到 if 语句时，执行的是 if 后面的语句，将 10 赋给 p 所指向的地址，并输出“将 10 赋值给 p 所指向的地址”。

实例 3.2.5　多级指针的使用

1. 问题描述与分析

指针变量是存放其他变量地址的变量，如果一个指针变量保存的是另一个指针变量的地址，称为指向指针的指针，或多级指针，可以定义二级指针、三级指针等。

在程序中定义变量、一级指针变量、二级指针变量，并为它们赋值，然后输出各种值。

2. 程序与运行结果

```
#include <stdio.h>
void main()
{
    int a, *p1, **p2;
    double b, *p3, **p4;
    a=10;
    b=22.3;
    p1=&a;
    p3=&b;
    p2=&p1;
    p4=&p3;
    printf("%d   %d   %d\n", a, *p1, **p2);
    printf("%4.1f   %4.1f   %4.1f\n", b, *p3, **p4);
    **p2=20;
    **p4=45.8;
    printf("%d   %d   %d\n", a, *p1, **p2);
    printf("%4.1f   %4.1f   %4.1f\n", b, *p3, **p4);
    printf("%X   %X   %X\n", &a, p1, *p2);
    printf("%X   %X\n", &p1, p2);
    printf("%X\n", &p2);
}
```

运行结果如下：

```
10   10   10
22.3   22.3   22.3
20   20   20
48.5   48.5   48.5
12FF7C   12FF7C   12FF7C
12FF78   12FF78
12FF74
```

调试程序，观察各个变量值的变化情况。

3. 知识点：多级指针

保存指针变量地址的变量，称为多级指针变量，保存一级指针变量地址的变量称为二级指针变量，保存二级指针变量地址的变量称为三级指针变量。

二级指针变量的定义格式如下：

```
数据类型 **指针变量名;
```

当然还可以定义三级指针变量、四级指针变量等，但一般情况下使用二级指针就可以解决问题了。

例如，本实例中定义的二级指针变量 p2 和 p4。p2 只能保存整型指针变量的地址，p4 只能保存双精度实型指针变量的地址。

程序定义了整型变量 a、整型指针 p1、二级整型指针 p2，以及双精度实型变量 b、双精度实型指针 p3、二级双精度实型指针 p4。

程序首先为变量 a 赋值为 10，为变量 b 赋值为 22.3，然后将 a 的地址赋给指针 p1，再将 p1 的地址赋给二级指针 p2。将 b 的地址赋给指针 p3，再将 p3 的地址赋给二级指针 p4。因此 * p1 就是 a 的值 10，* p2 就是 p1 的值(a 的地址)，**p2 就是 a 的值 10，因此第 1 行输出 3 个 10，同样第 2 行输出 3 个 22.3。然后使用“**p2=20;”和“**p4=45.8;”的方式为变量 a 和变量 b 赋值，再使用 3 种形式输出变量 a 和变量 b 的值。最后两行输出 a 的地址和 p1 的地址，a 的地址可以表示为 &a、p1 或 * p2，p1 的地址可以表示为 &p1 或 p2。

下面用图 3.6 表示变量 a、p1、p2 之间的关系(图中的每个单元表示 4B 的内存)，其中图 3.6(a)是开始时变量 a 的值是 10 的状态，图 3.6(b)是将变量 a 的值改变为 20 的状态。

地址	(a)	
0x12ff74	12ff78	p2
0x12ff78	12ff7c	p1
0x12ff7c	10	a

地址	(b)	
0x12ff74	12ff78	p2
0x12ff78	12ff7c	p1
0x12ff7c	20	a

图 3.6 各变量之间的关系

实例 3.2.6 动态分配内存

1. 问题描述与分析

使用动态内存分配技术，可以在程序运行过程中，当需要存储空间时，就马上分配内存，数据处理完之后，不再需要这些内存时，再将其释放，C 语言是通过几个函数来实现动态内存分配的。

程序中定义两个指针变量，然后为其动态分配内存，使用完后将内存释放。

2. 程序与运行结果

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int *p1;
    double *p2;
    printf("%X   %X\n", p1, p2);
    p1=(int *) malloc(sizeof(int));
    *p1=10;
    p2=(double *) malloc(sizeof(double));
    *p2=23.6;
```

```
    printf("%X  %d\n", p1, * p1);
    printf("%X  %4.1f\n", p2, * p2);
    printf("%X  %X\n", &p1, &p2);
    free(p1);
    free(p2);
}
```

运行结果如下：

```
CCCCCCCC  CCCCCCCC
431E70   10
431E30   23.6
12FF7C   12FF78
```

调试程序，观察变量 p1、p2 值的变化。

3. 知识点

1）malloc()函数

malloc()函数的原型如下：

```
void * malloc(unsigned int num_bytes);
```

其功能是在内存的动态存储区中分配一块长度为 num_bytes 字节的连续区域。如果分配成功，则返回指向被分配内存的指针(此存储区中的初始值不确定)，否则返回空指针 NULL。使用 malloc()函数需要包含头文件 stdlib.h。

在使用 malloc()函数动态申请内存时，一般需要将返回的指针强制转换成人们所需要的类型，例如，本实例中将第一次分配的内存指针强制转换为整型指针，再赋给 p1，将第二次分配的内存指针强制转换为双精度实型指针，再赋给 p2。

在刚定义完指针变量时，由于还没有具体指向，输出指针变量的结果都是 CCCCCCCC，用 malloc()函数分配内存后，两个指针变量就有了具体的值，然后就可以在动态分配的内存中赋值了。程序中的指针变量与动态分配内存的关系如图 3.7 所示。

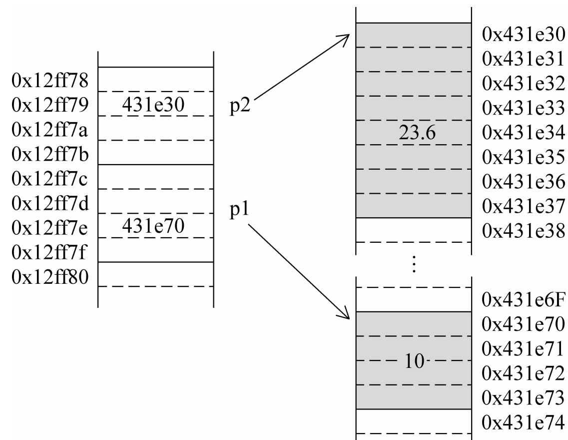

图 3.7　指针变量与其指向的内存单元

在程序中，人们只能去操作已经分配的内存单元，例如，只能在 0x431e70～0x431e73 这

4B处存取一个整数,或在0x431e30~0x431e36这8B处存取一个双精度实数。也就是说,只有这两块内存是由人们管理和使用的,人们可以随意保存或读取数据,如果想使用更多的内存,需要继续使用内存分配函数分配内存。

2) free()函数

free()函数的原型如下:

```
free(void * p);
```

作用是用来释放程序动态申请的内存,其参数是个指针类型。在C语言当中,只有当使用了动态内存申请函数malloc()、calloc()或realloc()申请内存之后,才可以使用free()函数来释放,释放之后就不能再使用了。

4. 知识补充

除malloc()函数可以动态分配内存外,还可以使用calloc()函数动态分配内存,calloc()函数的原型如下:

```
void * calloc(unsigned n,unsigned size);
```

功能是在内存的动态存储区中分配n个长度为size的连续空间,函数返回一个指向分配起始地址的指针,如果分配不成功,返回NULL。

calloc()函数与malloc()函数的区别是:calloc在动态分配完内存后,自动初始化该内存空间为零,而malloc不初始化,里边数据是随机的。通过下面的例子可以验证两者的区别。

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int * p1, * p2;
    p1=(int * )  malloc(sizeof(int));
    p2=(int * )  calloc(1,sizeof(int));
    printf("%x  %d\n", p1, * p1);
    printf("%x  %d\n", p2, * p2);
     * p1=10;
     * p2=20;
    printf("%x  %d\n", p1, * p1);
    printf("%x  %d\n", p2, * p2);
    free(p1);
    free(p2);
}
```

运行结果如下:

```
431e70  -842150451
431e40  0
431e70  10
431e40  20
```

实例 3.2.7　动态分配多个连续的内存单元

1. 问题描述与分析

在前面的实例中动态分配了一个数据单元，还可以利用这些函数动态分配多个连续的内存单元。

在程序中定义指针变量，分配多个连续的存储单元，然后为这些存储单元赋值，最后输出这些单元存放的数据，以及单元的地址。

2. 程序与运行结果

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int *p1;
    int *p2;
    p1=(int *)  malloc(3*sizeof(int));
    p2=(int *)  calloc(3, sizeof(int));
    printf("%d  %d  %d\n", *p1, *(p1+1), *(p1+2));
    printf("%d  %d  %d\n", *p2, *(p2+1), *(p2+2));
    *p1=10;
    *(p1+1)=20;
    *(p1+2)=30;
    *p2=40;
    *(p2+1)=50;
    *(p2+2)=60;
    printf("%d  %d  %d\n", *p1, *(p1+1), *(p1+2));
    printf("%d  %d  %d\n", *p2, *(p2+1), *(p2+2));
    printf("%x  %x\n", &p1, &p2);
    printf("%x  %x  %x\n", p1, p1+1, p1+2);
    printf("%x  %x  %x\n", p2, p2+1, p2+2);
    free(p1);
    free(p2);
}
```

运行结果如下：

```
-842150451  -842150451  -842150451
0  0  0
10  20  30
40  50  60
12ff7C  12ff78
431e60  431e64  431e68
431e20  431e24  431e28
```

3. 知识点：动态分配多个存储单元

calloc()函数和 malloc()函数都可以分配多个连续的存储单元，从程序的运行结果可以

发现，malloc()函数分配的内存单元没有被初始化，而 calloc()函数分配的内存单元都被初始化为 0。赋值之后的存储情况可用图 3.8 表示(图中每个单元表示 4B 的内存)。

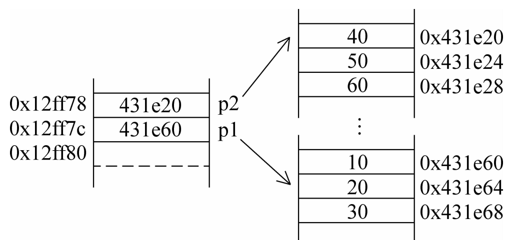

图 3.8 指针变量与其指向的连续多个内存单元

实例 3.2.8 重新分配存储单元

1. 问题描述与分析

动态分配一定的内存单元后，如果需要增加内存单元的数量，还可以使用 realloc()函数重新分配内存单元。

在程序中首先动态分配保存 3 个整数的存储单元，处理完一些数据后，再重新分配存储单元使其能够保存 50 个整型数据。

2. 程序与运行结果

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int *p1;
    int *p2;
    p1=(int *) malloc(3*sizeof(int));
    p2=(int *) malloc(3*sizeof(int));
    *p1=10;
    *(p1+1)=20;
    *(p1+2)=30;
    *p2=40;
    *(p2+1)=50;
    *(p2+2)=60;
    printf("%d  %d  %d\n", *p1, *(p1+1), *(p1+2));
    printf("%d  %d  %d\n", *p2, *(p2+1), *(p2+2));
    printf("%x  %x  %x\n", p1, p1+1, p1+2);
    printf("%x  %x  %x\n", p2, p2+1, p2+2);
    free(p1);
    p1=(int *) malloc(50*sizeof(int));
    p2=(int *) realloc(p2, 50*sizeof(int));
    printf("%d  %d  %d\n", *p1, *(p1+1), *(p1+2));
    printf("%d  %d  %d\n", *p2, *(p2+1), *(p2+2));
    printf("%x  %x  %x\n", p1, p1+1, p1+2);
```

```
    printf("%x  %x  %x\n", p2, p2+1, p2+2);
    free(p1);
    free(p2);
}
```

运行结果如下：

```
10  20  30
40  50  60
431e60  431e64  431e68
431e20  431e24  431e28
-842150451  -842150451  -842150451
40  50  60
431d20  431d24  431d28
431c20  431c24  431c28
```

调试程序，观察 * p1、*(p1+1)、*(p1+2)、* p2、*(p2+1)、*(p2+2)的值。在程序调试过程中不仅可以观察变量的值，还可以观察表达式的值。

3. 知识点：realloc()函数

realloc()函数的原型如下：

```
void * realloc(void * mem_address, unsigned int newsize);
```

功能是先判断当前的指针所指位置是否有足够的连续空间，如果有空间，扩大 mem_address 指向的地址，并且将 mem_address 返回；如果空间不够，先按照 newsize 指定的大小分配空间，将原有数据从头到尾复制到新分配的内存区域，而后释放原来 mem_address 所指内存区域，同时返回新分配的内存区域的首地址，即重新分配存储器块的地址。如果重新分配成功则返回指向被分配内存的指针，否则返回空指针 NULL。

例如，程序中首先分别为指针 p1 和 p2 动态分配 3 个存储整数的内存单元，然后给这些空间赋值并分别输出这些单元的值和地址。

在增加动态内存单元时，对指针 p1 和 p2 采用两种不同的方式。对于 p1，先用 free()函数释放原来的内存，再用 malloc()函数分配 50 个存储整数的单元，由于先释放再分配，因此原来的数据不能保存下来，3 个单元的值都变成了－842150451。对于 p2，使用 realloc()函数重新分配 50 个存储整数的单元，因此原来的数据被保存下来，仍然是 40、50、60。

3.3 小　　结

3.3.1 地址、指针与指针变量

1. 内存地址

计算机的内存储器被划分成一个个的存储单元，这些存储单元按一定的规则编号，这个编号就是存储单元的地址，地址编码的最基本单位是字节。

2. 变量的地址

在程序中定义的所有变量，在内存中都要分配相应的存储单元，系统分配给变量内存空

间的起始单元地址称为该变量的地址,变量的地址也称为该变量的指针。

3. 指针变量

用于存储其他变量的指针(地址)的变量,称为指针变量,指针变量的定义格式如下:

```
数据类型 *指针变量名;
```

可以通过运算符 * 和 & 分别获取指针所指向单元的内容或获取变量的地址。用于存储其他指针变量的指针(地址)的变量称为二级指针,二级指针变量的定义格式为

```
数据类型 **指针变量名;
```

同样也可以定义三级指针、四级指针。

3.3.2 指针的运算

由于指针变量存放的是地址,因此指针运算就是地址的运算。指针运算主要有算术运算和关系运算。

1. 指针的算数运算

指针可以与整数进行加减运算,指针与整数的加减运算结果与该指针所指向的数据类型有关。

如果 p 是一个指针变量,p+n 表示指针 p 当前所指向位置后面第 n 个数据的地址,p-n 表示指针 p 当前所指向位置前面第 n 个数据的地址。

指针变量也可以进行自增自减运算,p++与 p=p+1 的作用相同,p--与 p=p-1 的作用相同。

2. 指针的关系运算

指向同一种数据类型的指针可以进行关系运算。如果两个相同类型的指针相等,表示这两个指针指向同一个地址。

另外,指针也可以与 0(NULL)进行比较运算,如果 p==0 成立,称 p 是一个空指针,即指针 p 还没有具体指向。

3.3.3 动态内存分配

C 语言中与动态内存分配有关的函数包括 malloc、calloc、realloc 和 free。其中 malloc 函数、calloc 函数用于动态内存分配,realloc 函数用于重新分配内存,free 函数用于释放已经分配的内存。malloc 函数与 calloc 函数的区别是 malloc 函数分配的内存不被初始化,而 calloc 函数分配的内存被初始化为 0。

当不再使用后,在程序中动态分配的内存一定要用 free 函数将内存释放。

3.4 实 验

3.4.1 实验目的

(1) 理解内存地址、变量地址和变量指针的概念。

(2) 掌握指针变量的定义及使用方法。

(3) 掌握指针的算术运算和关系运算。

(4) 掌握动态分配内存的方法。

3.4.2 实验内容

实验 3.1 输入 a 和 b 两个数,按从小到大的顺序输出

1. 实验题目

在程序中定义两个整型变量 a 和 b,两个整型指针变量 p1 和 p2,通过键盘输入两个整数分别赋给 a 和 b,再将 a 的地址赋给 p1,将 b 的地址赋给 p2,然后通过对指针变量 p1 和 p2 的操作,实现按从小到大的顺序输出两个数。

2. 参考程序

```
#include <stdio.h>
void main()
{
    int a, b;
    int *p1, *p2, *p;
    printf("请输入两个整数: ");
    scanf("%d %d", &a, &b);
    p1=&a;
    p2=&b;
    if(*p1>*p2 )
    {
        p=p1;
        p1=p2;
        p2=p;
    }
    printf("min=%d  max=%d\n", *p1, *p2);
}
```

运行结果如下:

```
请输入两个整数: 65  23
min=23  max=65
```

其中第一行的 65 和 23 是程序运行时从键盘输入的。

程序首先从键盘上为变量 a 和 b 输入初值(假设输入 65 和 23),再使指针 p1 和 p2 分别指向 a 和 b,如图 3.9(a)所示。

在 if 语句中,其条件是 *p1> *p2,即 p1 所指向的单元内容如果大于 p2 所指向的单元的内容,就执行 if 后面的语句,因为 65 大于 23,所以执行 if 后面的语句。实际交换的是指针变量 p1 和 p2 本身的值,即指针 p1 指向了变量 b,指针 p2 指向了变量 a,而 a 和 b 的值并没有改变,如图 3.9(b)所示。在最后输出时,是先输出 *p1(即 b,23),后输出 *p2(即 a,65)。

单步运行程序,观察 p1 和 p2 值的变化。

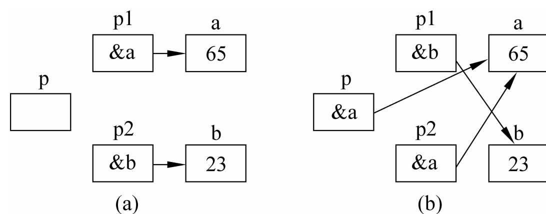

图 3.9 指针变量的内容交换前后的情况

实验 3.2 不定义变量,直接用两个指针完成实验 3.1 的功能

1. 实验题目

定义两个整型指针变量,并为它们动态分配内存,然后通过键盘输入两个整数,分别赋给前面动态分配的内存单元中,最后比较两个单元中数据的大小,先输出小的,后输出大的。

2. 参考程序

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int *p1, *p2;
    p1=(int *) malloc(sizeof(int));
    p2=(int *) malloc(sizeof(int));
    scanf("%d  %d", p1, p2);
    if(*p1> *p2)
        printf( "min=%d  max=%d\n", *p2, *p1);
    else
        printf( "min=%d  max=%d\n", *p1, *p2);
    free(p1);
    free(p2);
}
```

实验 3.3 用指针动态分配内存处并理学生成绩

1. 实验题目

在程序中定义指针变量,首先分配保存 5 个整数的内存空间,输入 5 个学生的成绩,再输出这 5 个学生的成绩,计算并输出平均成绩。然后又来了 5 个学生,重新分配保存 10 个整数的内存空间,输入新来 5 个学生的成绩,再输出这 10 个学生的成绩,最后计算并输出 10 个学生的平均成绩。

2. 参考程序

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
```

```
    int i;
    double sum=0;
    int *p;
    p=(int *) malloc(5 * sizeof(int));
    printf("请输入5个成绩：");
    for(i=0; i<5; i++)
    {
        scanf("%d", p+i);
    }
    printf( "5个学生的成绩是：");
    for(i=0; i<5; i++)
    {
        printf("%d  ", *(p+i));
    }
    printf( "\n");

    for(i=0; i<5; i++)
    {
        sum+=*(p+i);
    }
    printf( "5个学生的平均成绩=%4.1f\n", sum/5);

    p=(int *) realloc(p, 10 * sizeof(int));
    printf("请再输入5个成绩：");
    for(i=5; i<10; i++)
    {
        scanf("%d", p+i);
    }
    printf( "10个学生的成绩是：");
    for(i=0; i<10; i++)
    {
        printf("%d  ", *(p+i));
    }
    printf( "\n");
    sum=0;
    for(i=0; i<10; i++)
    {
        sum +=*(p+i);
    }
    printf( "10个学生的平均成绩=%4.1f\n", sum/10);
    free(p);
}
```

习　　题

3.1 若有定义“int x，*pb;”,则以下正确的赋值表达式是________。

A. pb=&x　　B. pb=x　　C. *pb=&x　　D. *pb=*x

3.2 执行语句“int i=10,*p=&i;”后，下面描述错误的是________。

A. p 的值为 10　　B. p 指向整型变量 i

C. *p 表示变量 i 的值　　D. p 的值是变量 i 的地址

3.3 执行以下两行语句后，不正确的赋值语句是________。

```
int a=5, b=10 , c;
int *p1=&a , *p2=&b;
```

A. *p2=b;　　B. p1=a;　　C. p2=p1;　　D. c= *p1 *(*p2);

3.4 若有以下两行语句，下面代表地址的有________。

```
int *point, a=4;
point=&a;
a,point,*&a,&*a,&a,*&point,*point
```

3.5 若有说明语句“int a, b, c, *d=&c;”，则能正确从键盘读入 3 个整数分别赋给变量 a、b、c 的语句是________。

A. scanf("%d %d %d", &a, &b, d);

B. scanf("%d %d %d", a, b, d);

C. scanf("%d %d %d", &a, &b, &d);

D. scanf("%d %d %d", a, b, *d);

3.6 写出以下程序的输出结果。

```
#include "stdio.h"
main()
{
    int *p,a;
    a=100;
    p=&a;
    a=*p+10;
    printf("%d\n",*p);
}
```